Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №4

по «Алгоритмам и структурам данных»

Базовые задачи (3 задачи)

Выполнил:

Студент группы P3210

Цыпандин Н. П.

Преподаватели:

Косяков М.С.

Тараканов Д. С.

Санкт-Петербург

2022

Задача №2 (N) «Свинки-копилки»

```
#include <iostream>
#include <vector>

using namespace std;

vector<vector<int>> graph;
vector<bool> opened;
vector<bool> used;

int dfs(int start) {
    int ans = 1;
    used[start] = true;
    for (int v: graph[start]) {
        if (!used[v] && !opened[v])
            ans += dfs(v);
        else if (!used[v])
            ans = ans + dfs(v) - 1;
    }
    return ans;
}

void clear_used(int n) {
    used.clear();
    used.resize(n, false);
}

int main() {
    int n, x;
    cin >> n;
    graph.resize(n);
    opened.resize(n, false);

    for (int i = 0; i < n; ++i) {
        cin >> x;
        graph[--x].push_back(i);
    }

    int ans = 0, opened_cnt = 0, best_v;
    while (opened_cnt < n) {
        int ma = 0;
        for (int i = 0; i < n; ++i) {
            if (opened[i])
                continue;

            clear_used(n);
            int res = dfs(i);
            if (res > ma) {
                ma = res;
                best_v = i;
            }
        }
        clear_used(n);
        opened_cnt += dfs(best_v);
        for (int i = 0; i < n; ++i) {
```

```
            if (used[i]) {
                opened[i] = true;
            }
        }
        ans++;
    }
    cout << ans << endl;
    return 0;
}
```

Пояснение к примененному алгоритму:

Представим в виде графа копилки. Ориентированное ребро проводится, если исходящая вершина - копилка, внутри которой ключ открыть копилку, в которое ребро входит. Далее, обойдем граф в глубину со всех вершин, и посчитаем какую компоненту связности нам удалось получить. Выберем макисмальную, и разобъем копилку - начальную вершину. Отметим всю компоненту как открытые, и будем повторять действия, не учитывая открытые копилки. Повторяем пока не получим доступ ко всем копилкам.

Асимптотика: O (n * n * n)

Т.к. мы проходимся обходом в глубину от каждой вершины для выбора максимума, dfs работает за O(V+E), вершин n, ребере n. Так же мы за один такой цикл выберем хотя бы 1 вершину, т.е. в худшем случае пройдемся n раз получая каждый раз 1 вершину.

Задача №3 (O) «Долой списывание!»

```
#include <iostream>
#include <vector>

using namespace std;

vector<vector<int>> graph;
vector<int> colors;

int get_other_color(int clr) {
    if (clr == 1)
        return 2;
    return 1;
}

void dfs(int start, int color) {
    colors[start] = color;
    for (int v: graph[start]) {
        if (colors[v] == 0) {
            dfs(v, get_other_color(color));
        } else if (colors[v] == color) {
            cout << "NO" << endl;
            exit(0);
        }
    }
}

int main() {
    int n, m;
    cin >> n >> m;
    graph.resize(n);

    int x, y;
    for (int i = 0; i < m; ++i) {
        cin >> x >> y;
        x--;
        y--;
        graph[x].push_back(y);
        graph[y].push_back(x);
    }

    for (int i = 0; i < n; ++i) {
        colors.clear();
        colors.resize(n, 0);
        dfs(i, 1);
    }

    cout << "YES" << endl;
    return 0;
}
```

Пояснение к примененному алгоритму:

Заметим, что задача сводится к проверке графа на двудольность (Группы списывающих и не списывающих). Проверим это с помощью раскраски поиском в глубину. Если получится раскрасить граф в 2 цвета, значит граф двудольный, значит можно разделить на 2 группы. Для этого попробуем раскрасить граф начиная с каждой вершины.

Асимптотика: O(N * (N + M))

Задача №4 (Р) «Авиаперелёты»

```
#include <iostream>
#include <vector>

using namespace std;

vector<vector<int>> graph;
vector<bool> used;
int USED_CNT = 0;

void dfs(int current, int limit) {
    used[current] = true;
    USED_CNT++;
    for (int i = 0; i < graph.size(); ++i) {
        if (i != current && !used[i] && graph[current][i] <= limit) {
            dfs(i, limit);
        }
    }
}

void dfs_backwards(int current, int limit) {
    used[current] = true;
    USED_CNT++;
    for (int i = 0; i < graph.size(); ++i) {
        if (i != current && !used[i] && graph[i][current] <= limit) {
            dfs_backwards(i, limit);
        }
    }
}

void restore_all() {
    used.clear();
    used.resize(graph.size());
    USED_CNT = 0;
}

bool check_limit(int limit) {
    bool straight = false, backwards = false;
    restore_all();
    dfs(0, limit);
    if (USED_CNT == graph.size())
        straight = true;

    restore_all();
    dfs_backwards(0, limit);
    if (USED_CNT == graph.size())
        backwards = true;

    if (backwards && straight)
        return true;
    return false;
}

int main() {
    int n;
    cin >> n;
    graph.resize(n);
    restore_all();

    int weight, ma = 0;
    for (int i = 0; i < n; ++i) {
```

```
        graph[i].resize(n);
        for (int j = 0; j < n; ++j) {
            cin >> weight;
            graph[i][j] = weight;
            if (weight > ma)
                ma = weight;
        }
    }

    int left = 0, right = 2 * ma, mid;
    while (left < right) {
        mid = (left + right) / 2;
        if (check_limit(mid)) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }
    cout << left << endl;
    return 0;
}
```

Пояснение к примененному алгоритму:

Решим задачу с помощью бинарного поиска по ответу. Для потенциального ответа проверим, можно ли долететь со всех вершин во все вершины. Для этого мысленно исключим все ребра, вес которых больше потенциального ответа и пройдемся поиском в глубину. Заметим, что граф ориентированный, относительно весов, так что будем запускать 2 поиска в глубину, один который проверяет пути только наизнанку. Тогда действительно получиться проверить на факт того, можно ли добраться с каждой вершины до каждой. Если мы для потенциального ответа получили в компоненте все вершины в 2 разных dfs, то сдвинем right.

Асимптотика: O(n^2)

Т.к. бинарный поиск за константу (log2(1e9)), 2 поиска в глубину, где ребер n*(n-1)/2 и количество вершин n, что пренебрежительно мало относительно n^2.